Benchmark Study on Distributed XML Filtering Using Hadoop Distribution Environment

Sanjay Kulhari, Jian Wen
UC Riverside

Team

Sanjay Kulhari
M.S. student, CS
U C Riverside

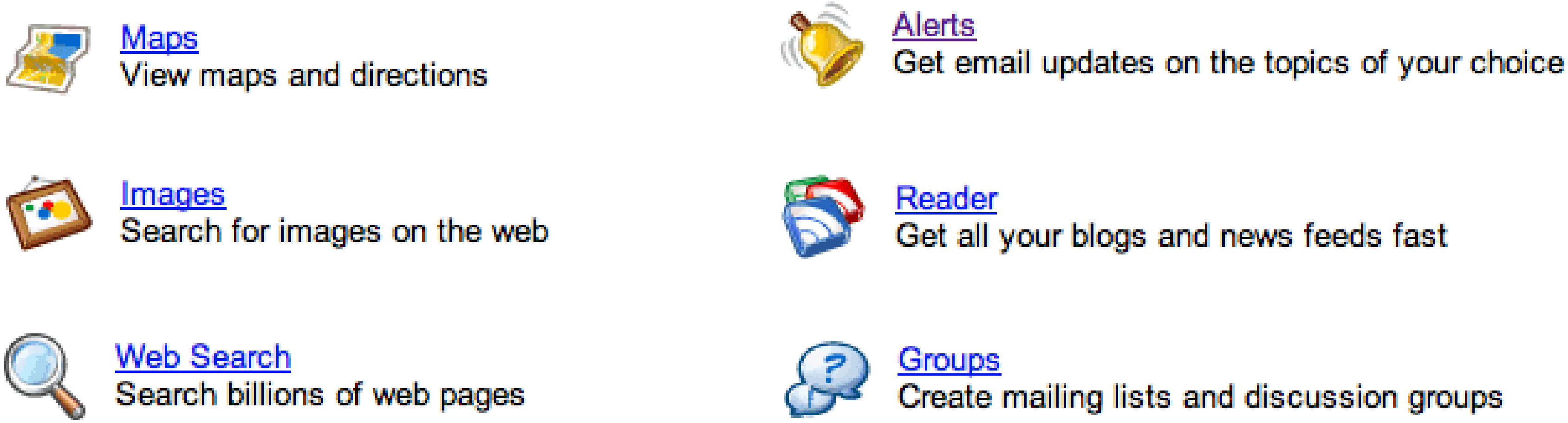
Jian Wen
Ph.D. student, CS
U C Riverside

Outline

- Pub/Sub Systems
- Project Overview and Goals
- Theoretical Core: XML Filtering
- Implementation:
  - single-threaded, multi-threaded, MR
- Experimental Evaluation
- Conclusions and Future Work.

Pub/Sub Systems: Motivation

- Think about the following Google's services



Maps
View maps and directions

Images
Search for images on the web

Web Search
Search billions of web pages

Alerts
Get email updates on the topics of your choice

Reader
Get all your blogs and news feeds fast

Groups
Create mailing lists and discussion groups

Input queries and get results!

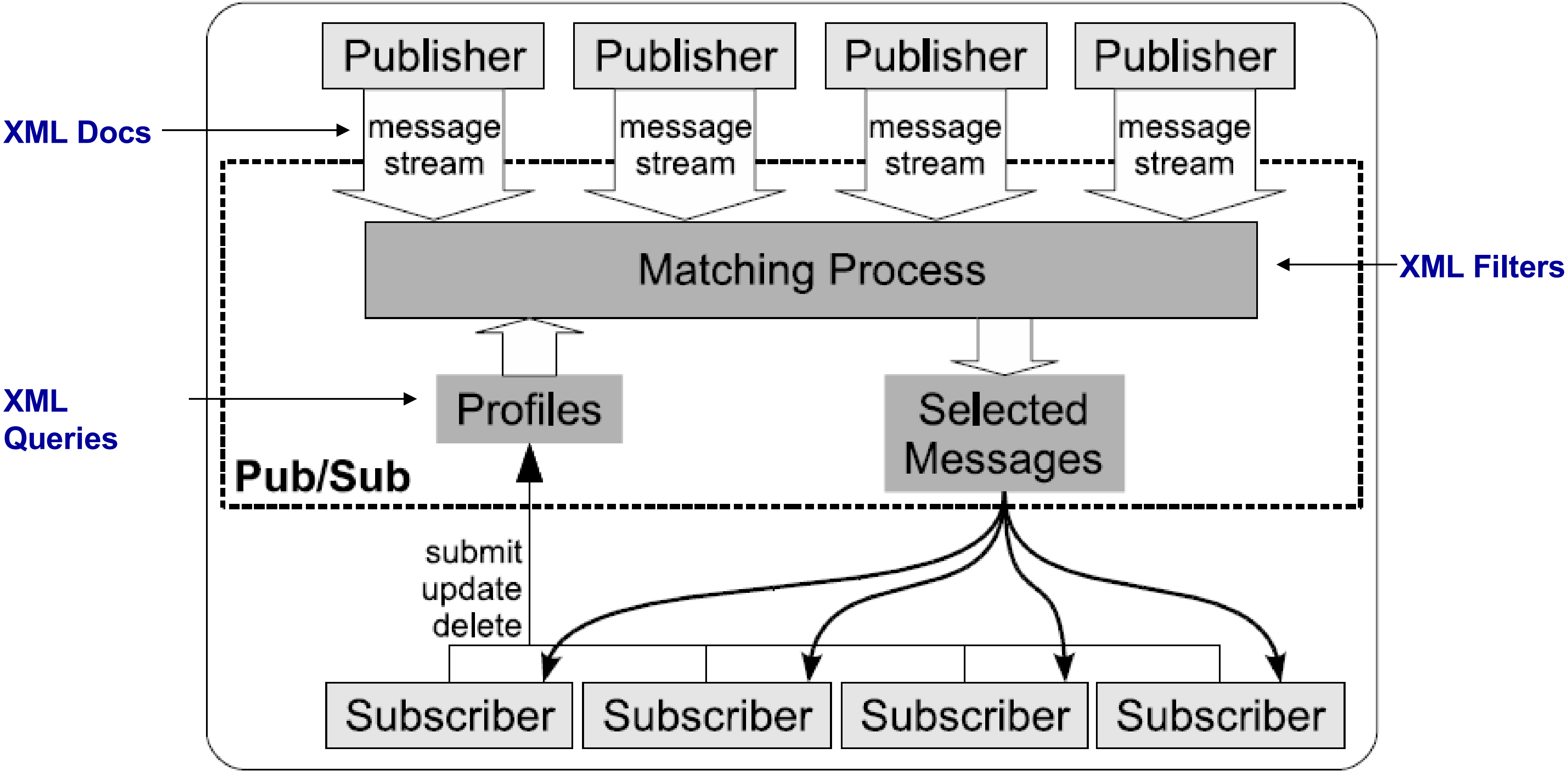
Register your interests and get updates!

Pub/Sub Systems: Properties

- Compared with traditional search services:

Traditional Query	Pub/Sub
Documents are known.	Queries are known.
Queries may come any time.	Documents are feed as a stream.
Users need quick answers. (always!)	Real time (a log monitoring system) or non-real time (Google Group…)

Architecture of a Pub-Sub System



Project Overview and Goals

- To provide distributed XML filtering with high scalability on large clusters.
  - XML filtering is a resource intensive operation.
  - Number of profiles to be matched can be huge.
  - Length of the profile can be huge.
- In our project, the scalability of the YFilter is checked.
  - Three benchmark platforms: single-threaded, multi-threaded, and map/reduce.
  - Goal: Any gains from distributing the algorithm?

Theoretical Core: XML Filtering

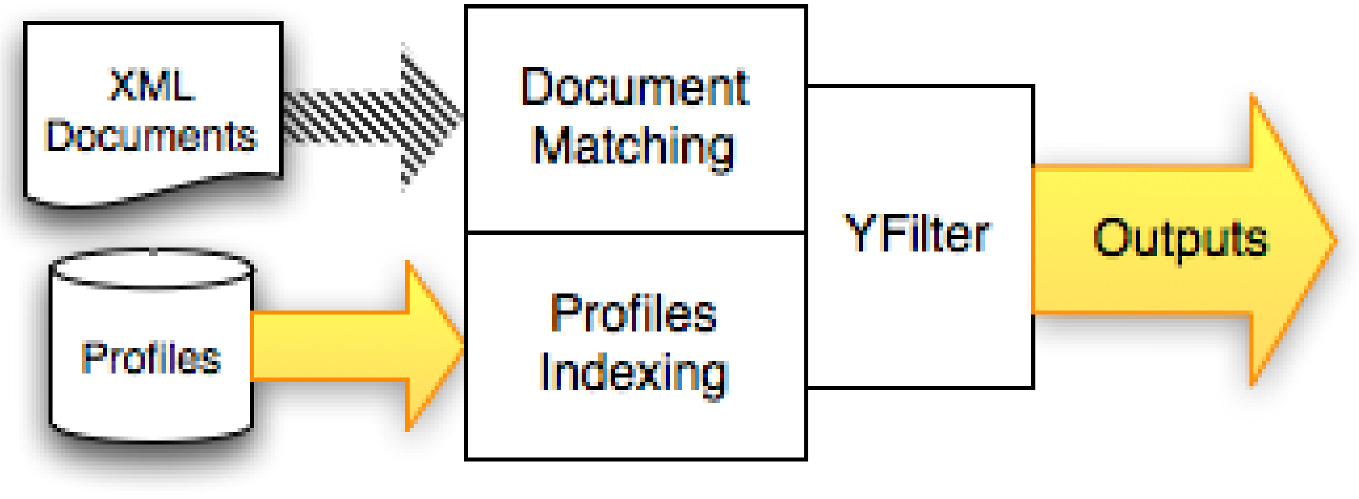
- Documents are matched to specified XPath queries
- Required for publish-subscribe systems
- Index is created on available subscription requests (XPath profiles)

Theoretical Core: Filtering Algorithms

- There are many existing works on filtering algorithms:
  - Software: Profiles are indexed (as finite state machine, for example).
  - Hardware: Profiles are mapped into FPGA devices.
- Our choice: YFilter
  - Parallel-able.
  - Efficient.
  - Easy to implement.

Theoretical Core: YFilter (Original)

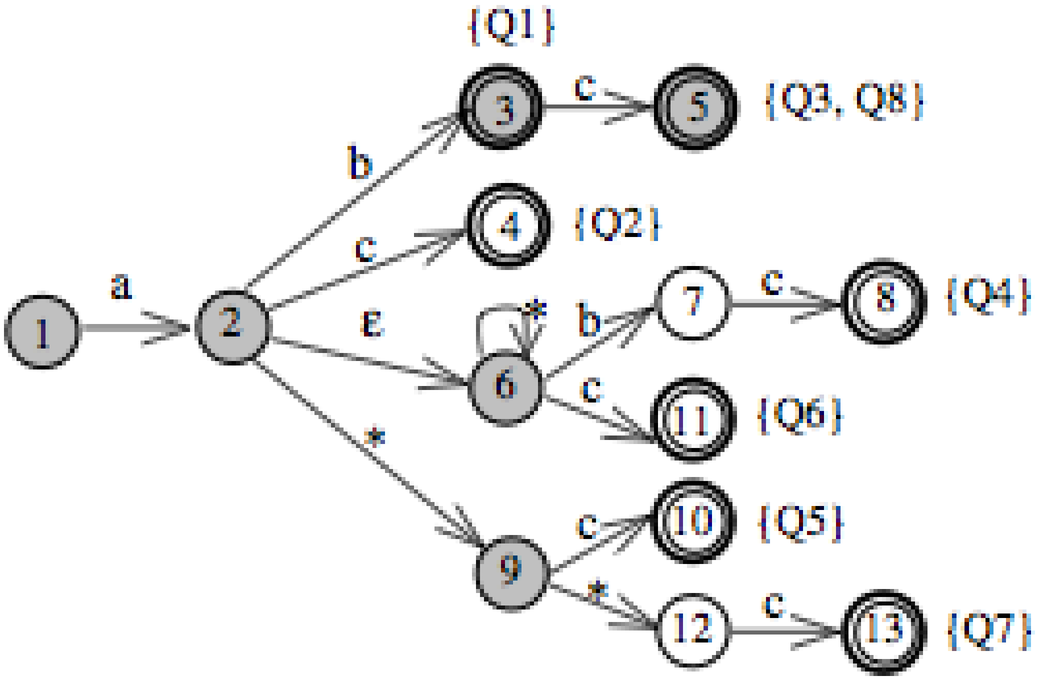
- Profiles are indexed as a NFA in advance.
- Documents then are fed into the filter.
- The matching query is processed by traversing the NFA.



Theoretical Core: YFilter (Original)

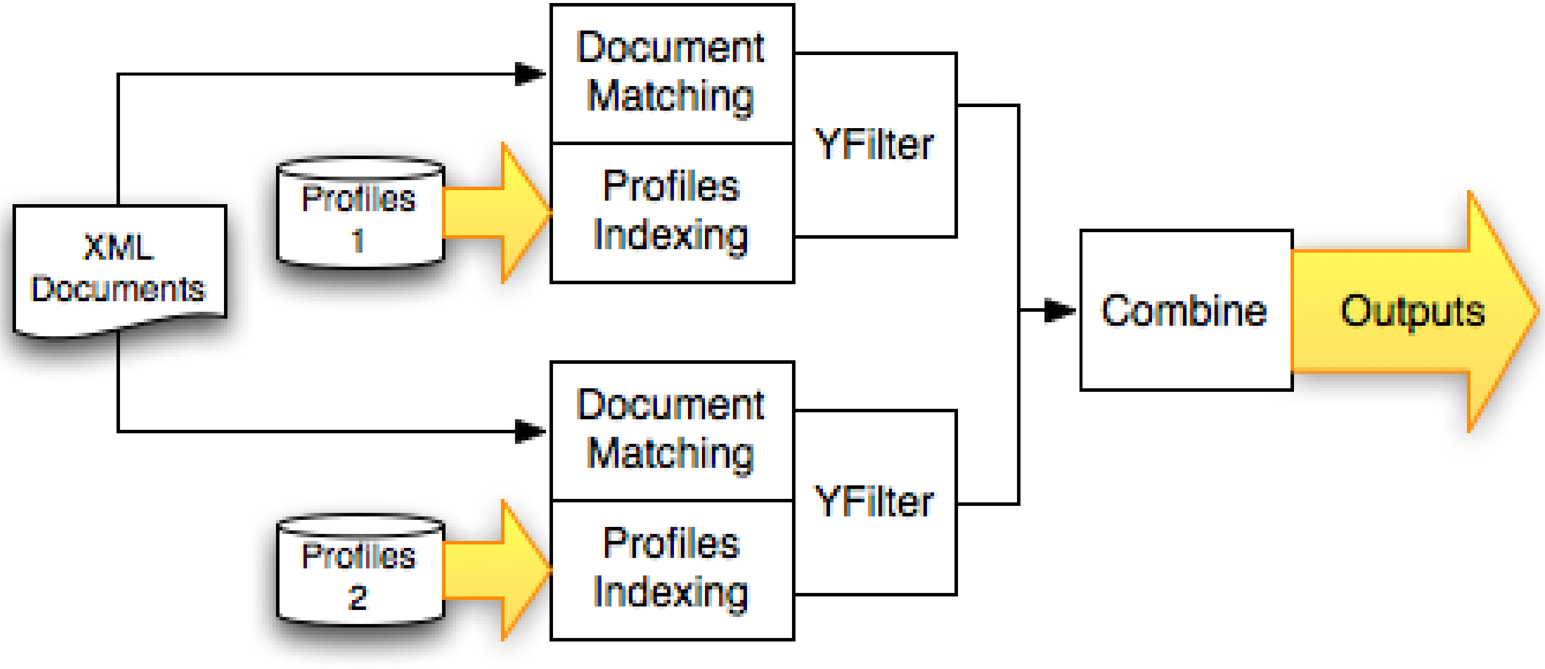
- NFA built in YFilter

Q1=/a/b
Q2=/a/c
Q3=/a/b/c
Q4=/a//b/c
Q5=/a/*/c
Q6=/a//c
Q7=/a/*/*/c
Q8=/a/b/c



Theoretical Core: YFilter (Parallel)

- YFilter is easy to be paralleled: profiles can be divided into parts and be indexed separately.

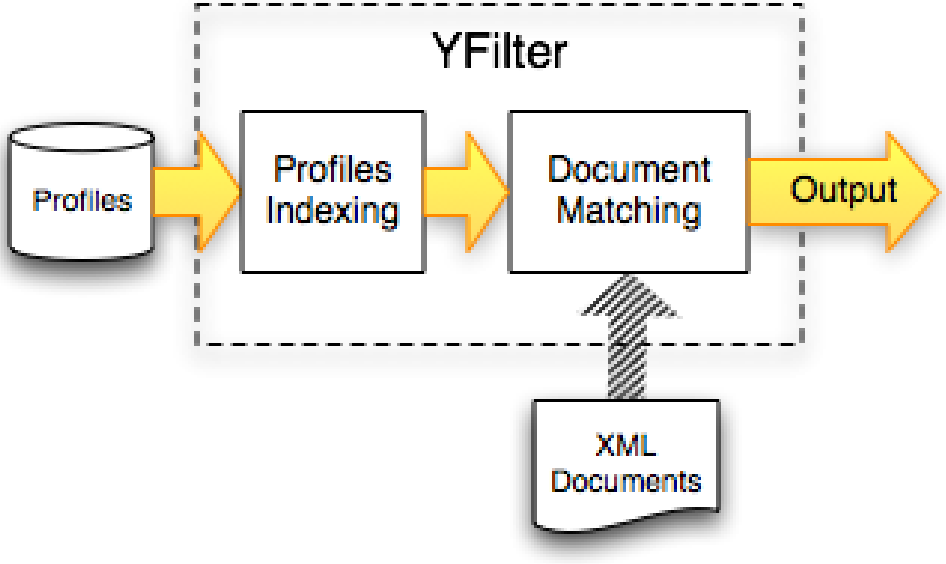


Project Implementations

- Three benchmark platforms are implemented in our project:
  - Single-threaded: Directly apply the YFilter on the profiles and document stream.
  - Multi-threaded: Parallel YFilter onto different threads.
  - Map/Reduce: Parallel YFilter onto different machines (currently in pseudo-distributed environment).

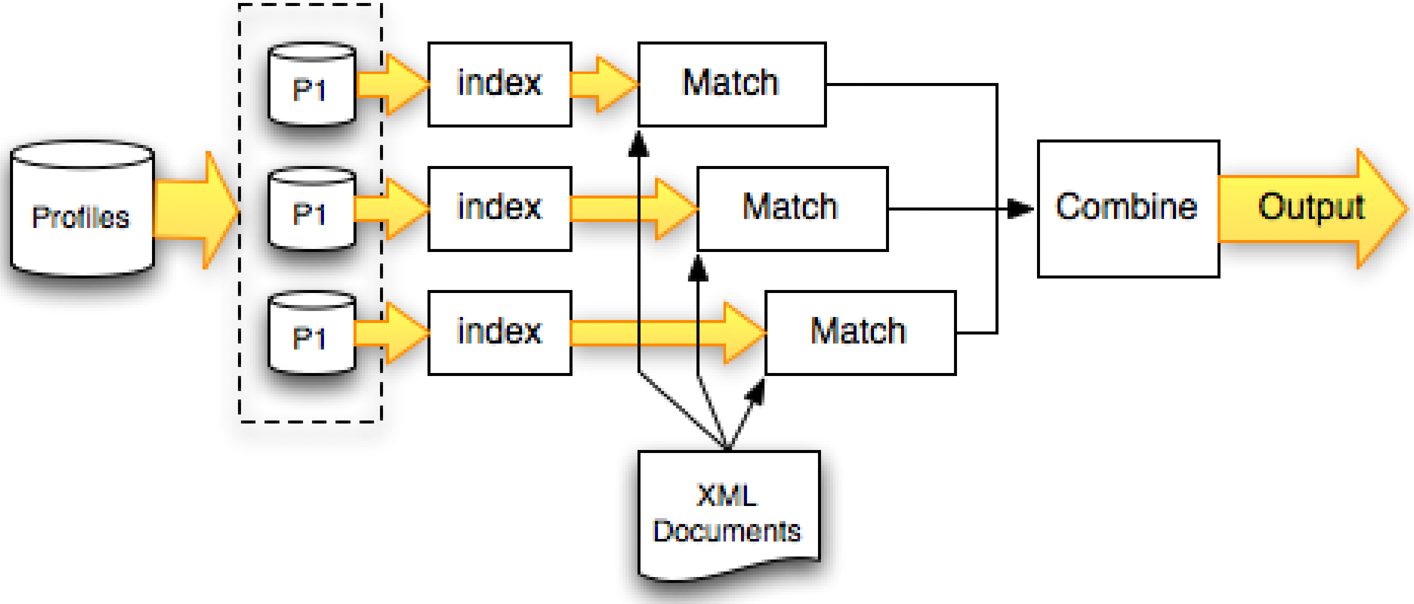
Benchmark 1: Single Thread

- The index (NFA) is built once on the whole set of profiles.
- Documents then are streamed into the YFilter for matching.
- Matching results then are returned by YFilter.



Benchmark 2: Multiple Threads

- Profiles are split into parts, and each part of the profiles are used to build a NFA separately.
- Each YFilter instance listens a port for income documents, then it outputs the results through the socket.



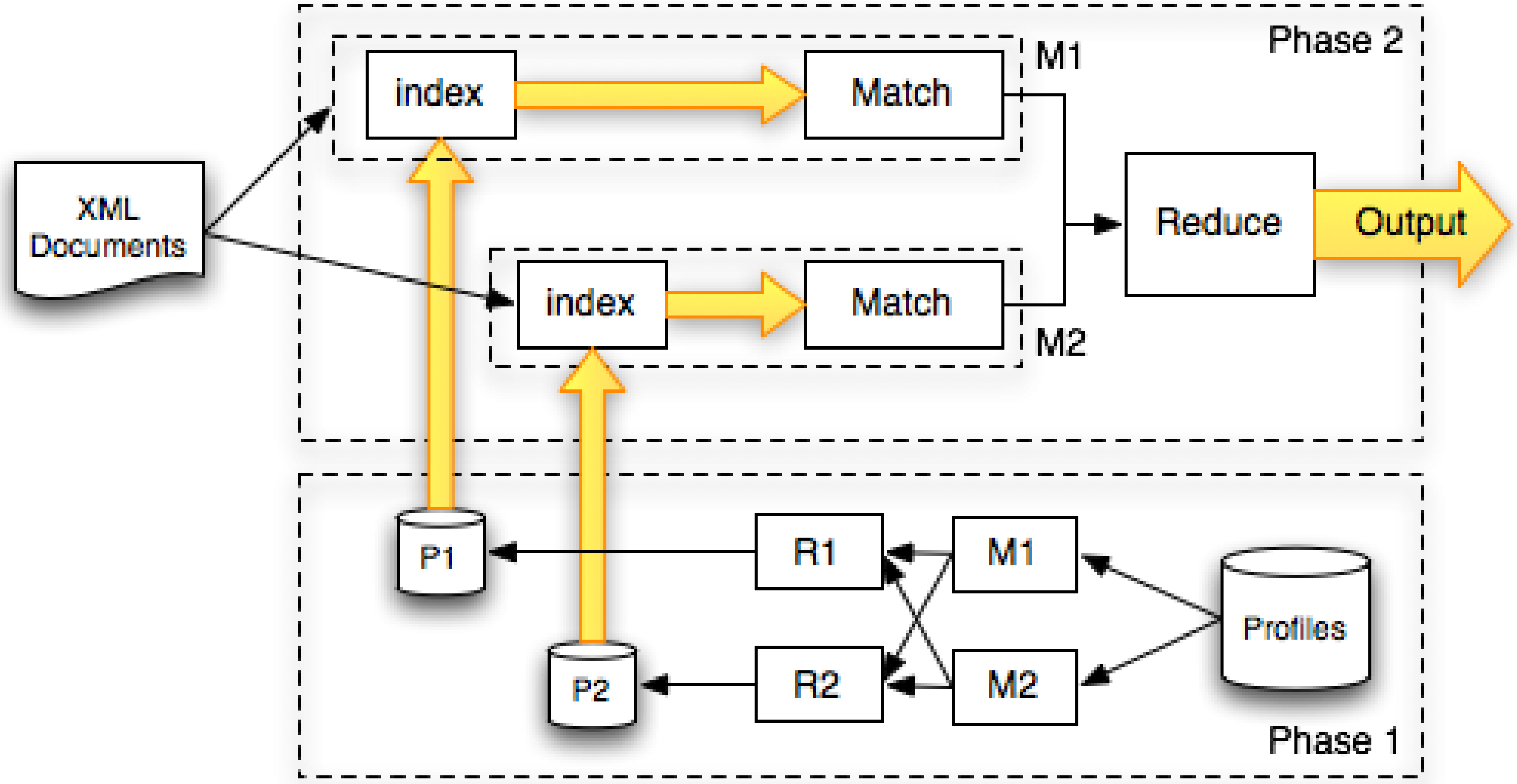
Benchmark 3: Map/Reduce

- Same strategy as the multi-threaded version, however all process are handled by Hadoop.
- Profile splitting: Profiles are read line by line with line number as the key and profile as the value.
  - Map: For each profile, assign a new key using (old_key % split_num)
  - Reduce: For all profiles with the same key, output them into a file.
  - Output: Separated profiles, each with profiles having the same (old_key % split_num) value.

Benchmark 3: Map/Reduce

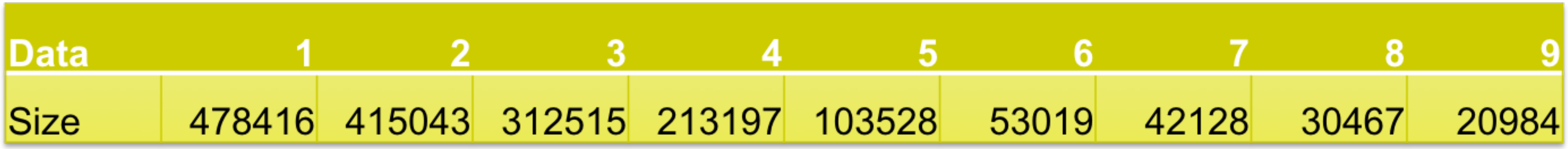
- Document matching: Split profiles are read file by file with file number as the key and profiles as the value.
  - Map: For each set of profiles, run YFilter on the document (fed as a configuration of the job), and output the old_key of the matching profile as the key and the file number as the values.
  - Reduce: Do nothing.
  - Output: All keys (line numbers) of matching profiles.

Benchmark 3: Map/Reduce



Experimental Evaluation

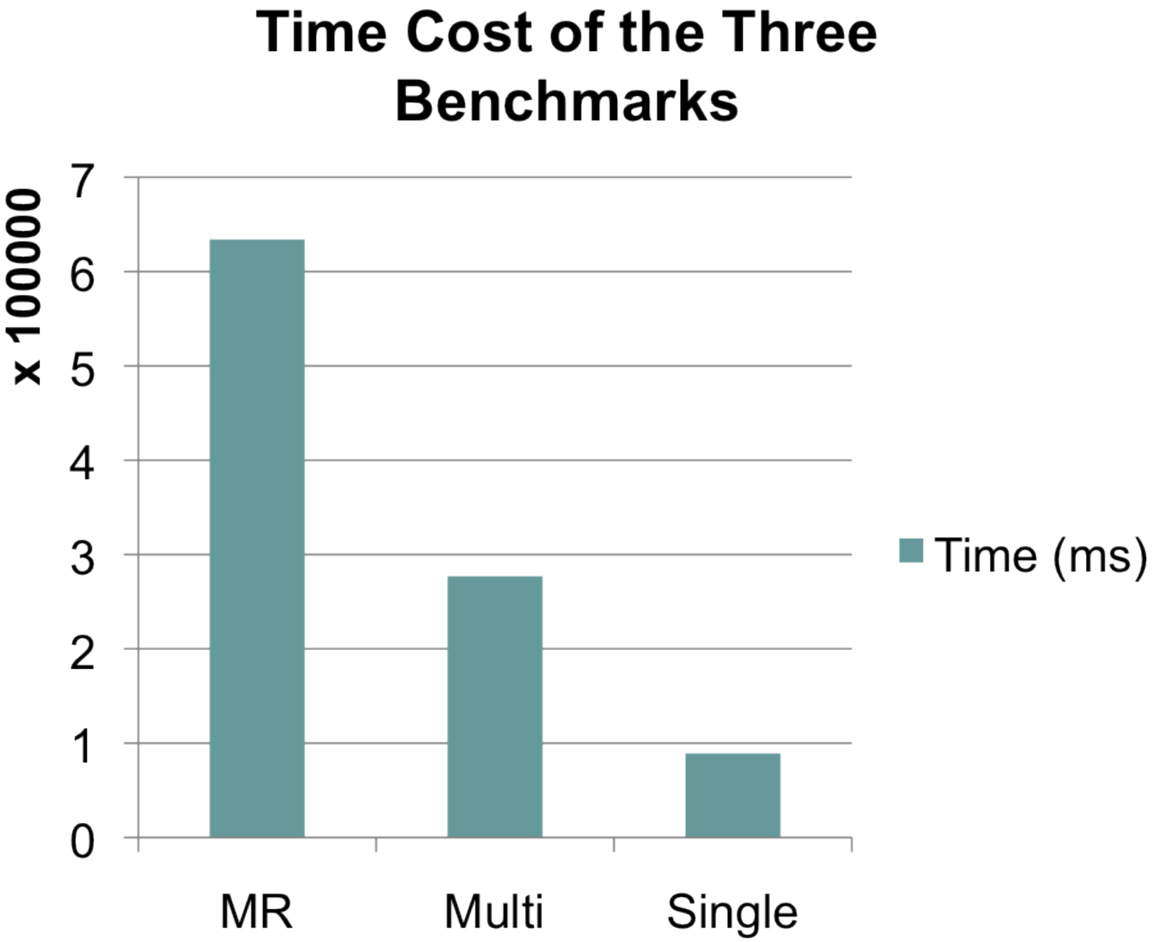
- Hardware:
  - Macbook 2.2 GHz Intel Core 2 Duo
  - 4G 667 MHz DDR2 SDRAM
- Software:
  - Java 1.6.0_17, 1GB heap size
  - Cloudera Hadoop Distribution (0.20.1) in a virtual machine.
- Data:
  - XML docs: SIGMOD Record (9 files).
  - Profiles: 25K and 50K profiles on SIGMOD Record.



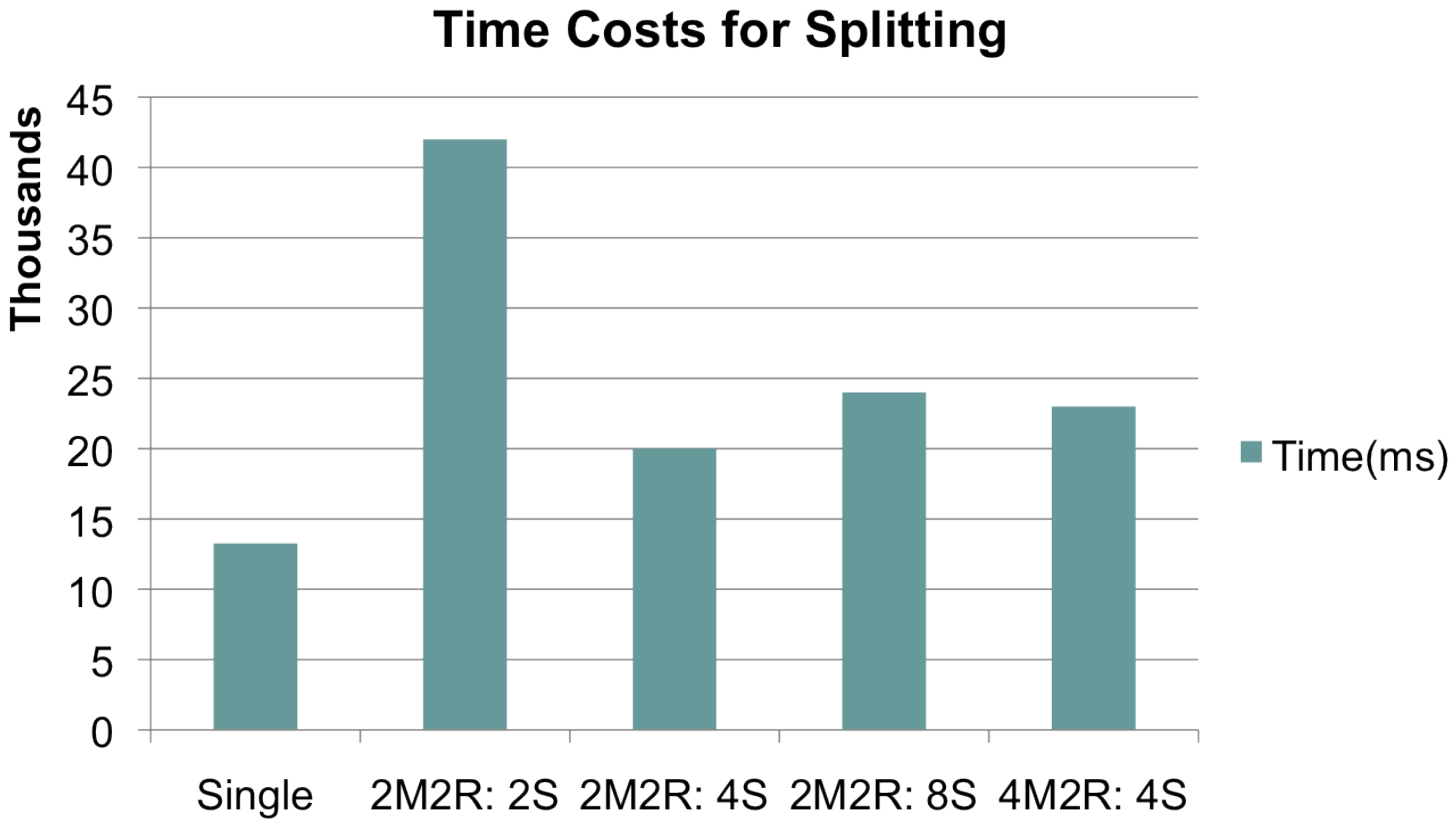
Data	1	2	3	4	5	6	7	8	9
Size	478416	415043	312515	213197	103528	53019	42128	30467	20984

Experimental Evaluation

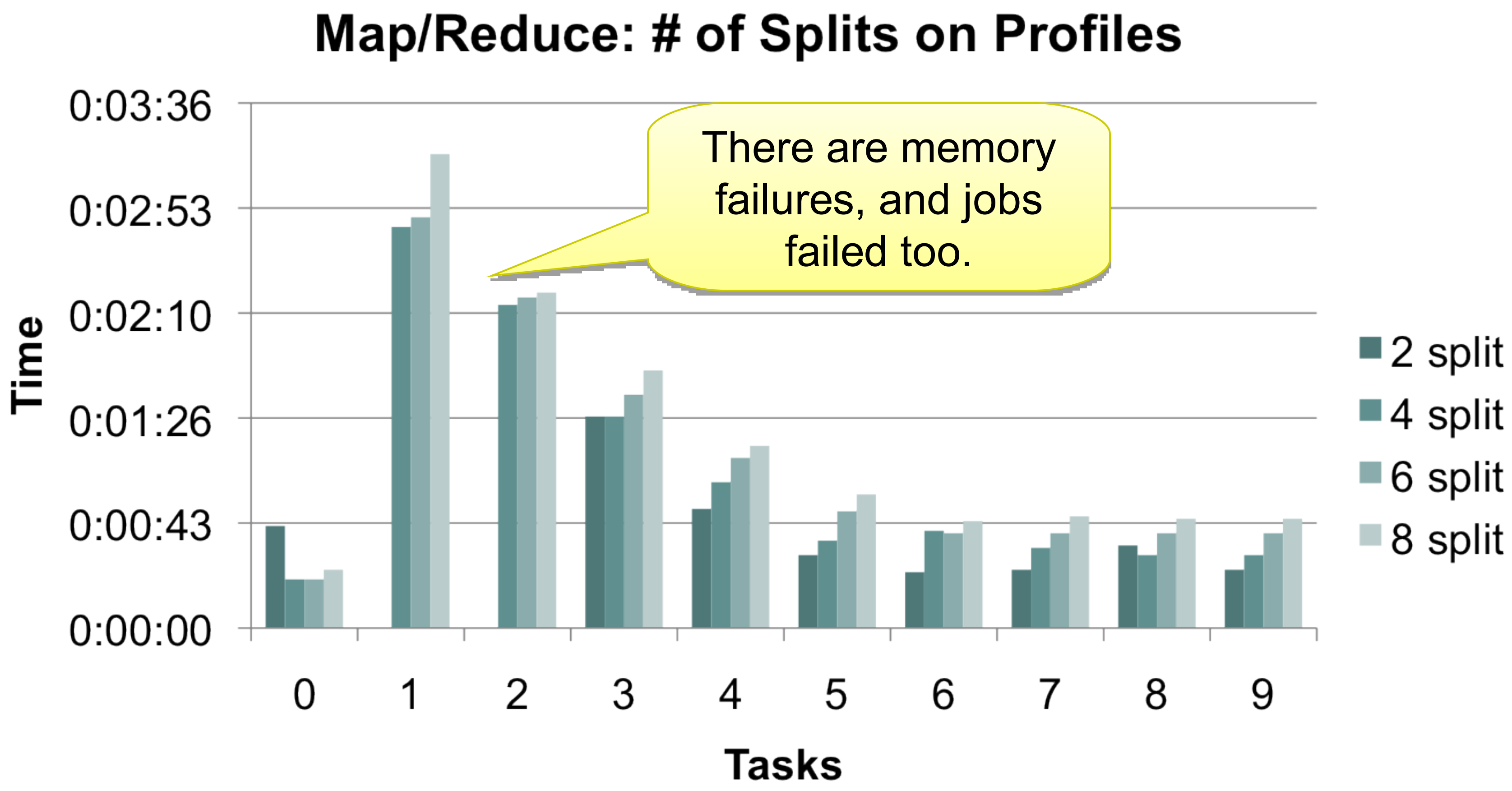
- Since all tests are now running on a single machine, any attempts on parallel may decrease the performance.
- Although the CPU is duo core, many administrative costs may decrease the performance significantly.



Experimental Evaluation

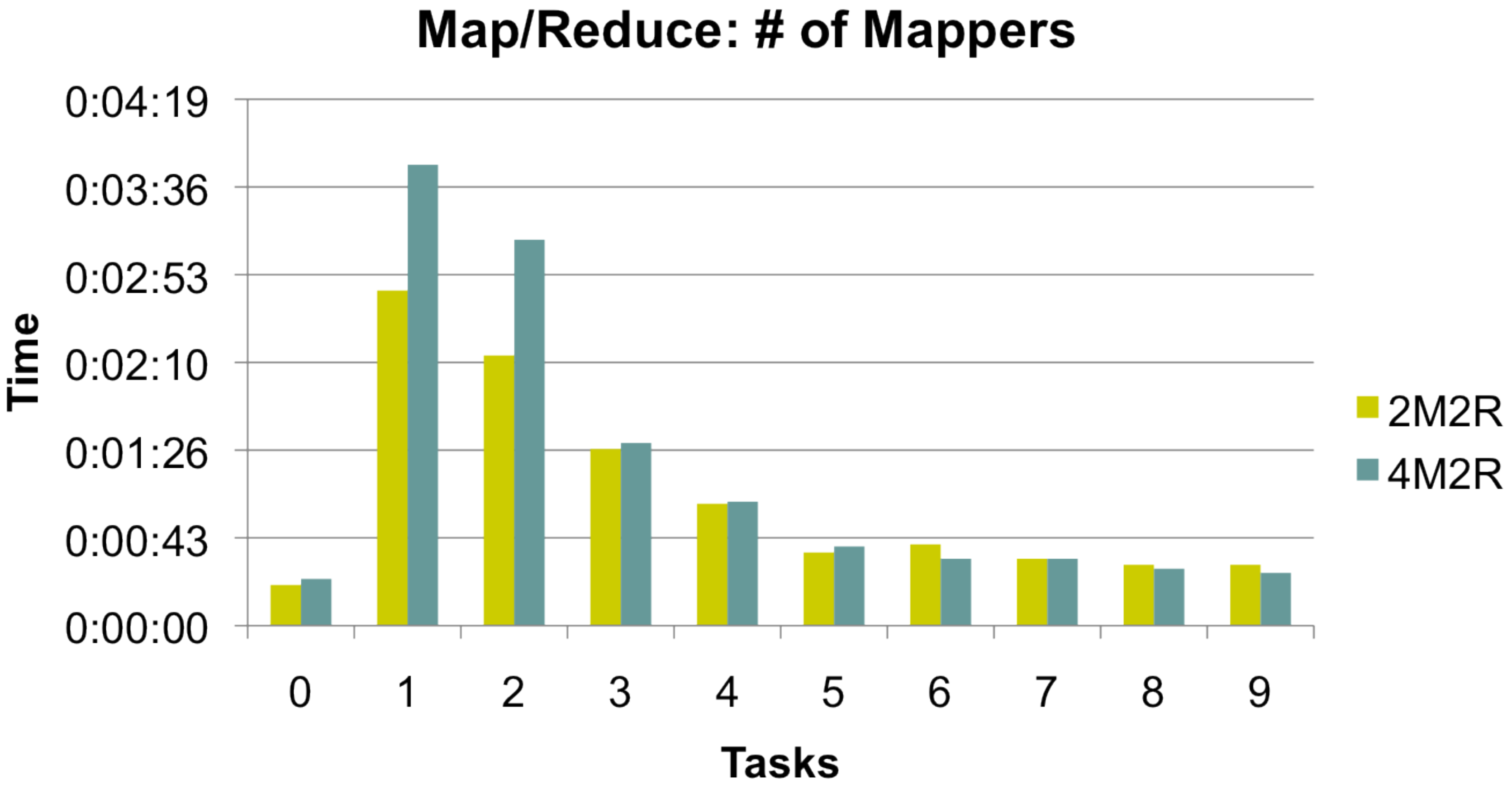


Experimental Evaluation

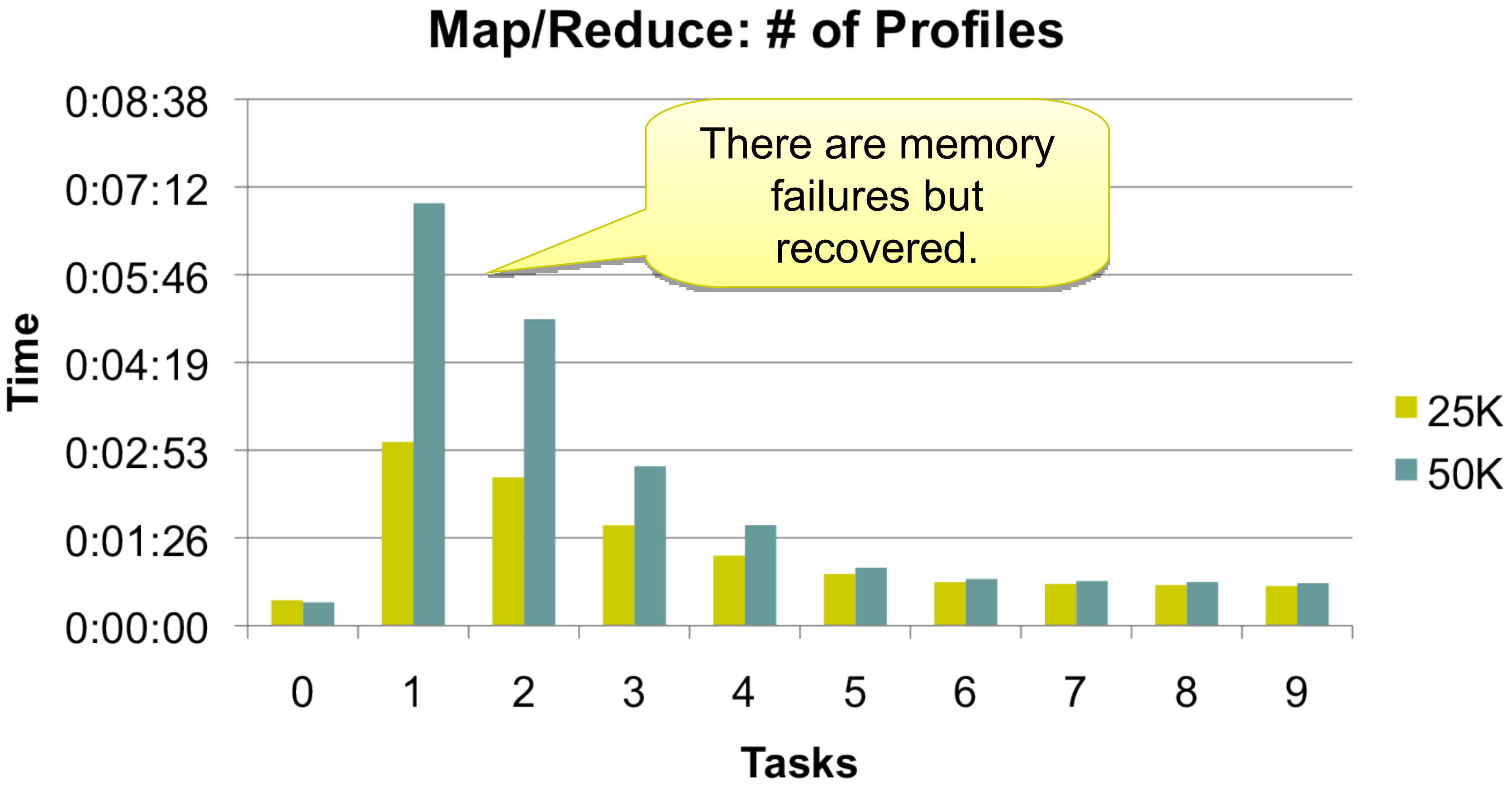


There are memory failures, and jobs failed too.

Experimental Evaluation



Experimental Evaluation



There are memory failures but recovered.

Interesting Stuffs

- Run-out-of-memory: We encountered this problem in all the three benchmarks, however Hadoop is much robust on this:
  - Smaller profile split
  - Map phase scheduler uses the memory wisely.
- Race-condition: since the YFilter code we are using is not thread-safe, in multi-threaded version race-condition messes the results; however Hadoop works this around by its shared-nothing run-time.
  - Separate JVM are used for different mappers, instead of threads that may share something lower-level.

Conclusion and Future Work

- Conclusion
  - XML pub/sub systems on large cluster is feasible.
  - Single machine tests show that no performance gains can be achieved by paralleled through threads/virtual machines.
  - Hadoop provides better framework on handling parallel and fault tolerance.
- Future Work
  - Tests on real distributed environment.
  - More inspection on the map/reduce framework for stream processing.

References

- 2002, ICDE '02: Proceedings of the 18th International Conference on Data Engineering, *YFilter: Efficient and Scalable Filtering of XML Documents. IEEE Computer Society, p.341.*
- Condie, T., Conway, N., Alvaro, P., Hellerstein, J.M., Elmeleegy, K. & Sears, R., 2009, MapReduce Online, *UC Berkley Technique Report.*
- YFilter: http://yfilter.cs.umass.edu/
- Cloudera Hadoop Distribution: http://www.cloudera.com/hadoop

Questions